How to build with Nano Banana: Complete Developer Tutorial

dev.to/googleai/how-to-build-with-nano-banana-complete-developer-tutorial-646

Patrick Loeber September 4, 2025

Google has recently released Gemini 2.5 Flash Image, a powerful new model for image generation and editing, also known by its codename, Nano Banana. This model introduces state-of-the-art capabilities for creating and manipulating images, unlocking a wide range of new applications.

This guide provides a comprehensive walkthrough for developers looking to integrate Gemini 2.5 Flash Image aka Nano Banana into their applications using the Gemini Developer API.

This guide will cover:

1. Using Nano Banana in AI Studio
2. Project setup
3. Image creation
4. Image editing
5. Photo restoration
6. Multiple input images
7. Conversational image editing
8. Best practices and effective prompting
9. Community examples and inspiration
10. Resources

Here's an example of what you'll build in this tutorial:

```
prompt = "Restore and colorize this image from 1932"

response = client.models.generate_content(
    model="gemini-2.5-flash-image-preview",
    contents=[prompt, image],
)
```

Let's get started!

If you'd prefer a video version of this post, you can watch it here:

Watch Video At: https://youtu.be/UTdfxFyOQTI

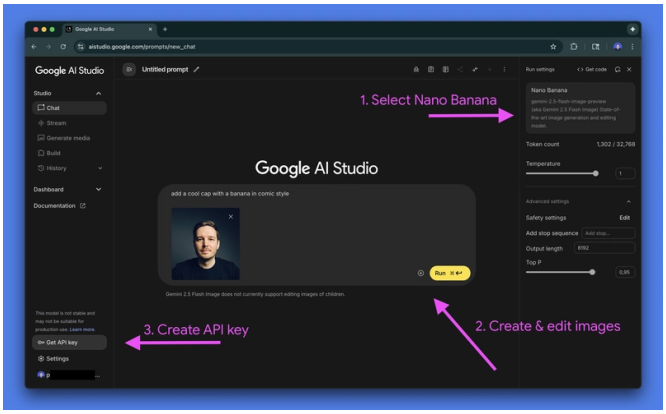
1) Using Nano Banana in Google AI Studio

While end-users can access Nano Banana in the Gemini app, the best environment for developers to prototype and test prompts is Google AI Studio. AI Studio is a playground to experiment with all available AI models before writing any code, and it's also the entry point for building with the Gemini API.

You can use Nano Banana free of charge within AI Studio. To get started, go to aistudio.google.com, sign in with your Google account, and select **Nano Banana** from the model picker.

For direct access, use this link to start a new session with the model:

ai.studio/banana



> **Tip**: You can also vibe code Nano Banana web apps directly in AI Studio at ai.studio/apps, or explore the code and remix one of the existing apps.

2) Project setup

To follow this guide, you will need the following:

- An API key from Google AI Studio.
- Billing set up for your project.
- The Google Gen AI SDK for Python or JavaScript/TypeScript.

Step A: Generate an API Key

Follow these steps:

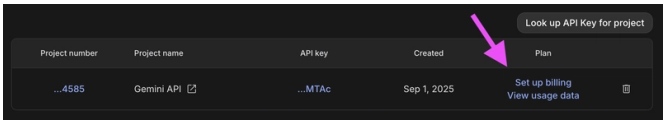
- In Google AI Studio, click **Get API key** in the left navigation panel.
- On the next page, click **Create API key**.
- Select an existing Google Cloud project or create a new one. This project is used to manage billing for API usage.

Once the process is complete, your API key will be displayed. Copy and store it securely.

Step B: Enable Billing

While prototyping in AI Studio is free, using the model via the API is a paid service. You must enable billing on your Google Cloud project.

In the API key management screen, click **Set up billing** next to your project and follow the on-screen instructions.



How much does Nano Banana cost?

Image generation with Nano Banana costs **$0.039 per image** *. For $1, you can generate approximately 25 images.

> * The official pricing is $0.30/1M input tokens and $30/1M output tokens. A standard 1024x1024px output image consumes 1290 tokens, which equates to $0.039 per image. For details, refer to the Gemini 2.5 Flash Image pricing table.

Step C: Install the SDK

Choose the SDK for your preferred language.

Python:

```
pip install -U google-genai
# Install the Pillow library for image manipulation
pip install Pillow
```

JavaScript / TypeScript:

```
npm install @google/genai
```

> The following examples use the Python SDK for demonstration. Equivalent code snippets to **use Nano Banana in JavaScript** are provided in this GitHub Gist.

3) Image Generation from Text

Use Nano Banana to generate one or more images from a descriptive text prompt. Use the model ID `gemini-2.5-flash-image-preview` for all API requests.

```
from google import genai
from PIL import Image
from io import BytesIO

# Configure the client with your API key
client = genai.Client(api_key="YOUR_API_KEY")

prompt = """Create a photorealistic image of an orange cat
with a green eyes, sitting on a couch."""

# Call the API to generate content
response = client.models.generate_content(
    model="gemini-2.5-flash-image-preview",
    contents=prompt,
)

# The response can contain both text and image data.
# Iterate through the parts to find and save the image.
for part in response.candidates[0].content.parts:
    if part.text is not None:
        print(part.text)
    elif part.inline_data is not None:
        image = Image.open(BytesIO(part.inline_data.data))
        image.save("cat.png")
```

Output:

The model is multimodal, so the response is structured as a list of `parts` that can contain interleaved text and image data (`inline_data`). The code above iterates through these parts to extract and save the generated image.

4) Image Editing with Text and Image Inputs

Provide an existing image along with a text prompt to perform edits. The model excels at maintaining character and content consistency from the input image.

```
from google import genai
from PIL import Image
from io import BytesIO

client = genai.Client(api_key="YOUR_API_KEY")

prompt = """Using the image of the cat, create a photorealistic,
street-level view of the cat walking along a sidewalk in a
New York City neighborhood, with the blurred legs of pedestrians
and yellow cabs passing by in the background."""

image = Image.open("cat.png")

# Pass both the text prompt and the image in the 'contents' list
response = client.models.generate_content(
    model="gemini-2.5-flash-image-preview",
    contents=[prompt, image],
)

for part in response.candidates[0].content.parts:
    if part.text is not None:
        print(part.text)
    elif part.inline_data is not None:
        image = Image.open(BytesIO(part.inline_data.data))
        image.save("cat2.png")
```

Input and Output:

5) Photo restoration with Nano Banana

One of the model's powerful applications is photo restoration. With a simple prompt, it can restore and colorize old photographs with impressive results.

```
from google import genai
from PIL import Image
from io import BytesIO

client = genai.Client(api_key="YOUR_API_KEY")

prompt = "Restore and colorize this image from 1932"

image = Image.open("lunch.jpg")  # "Lunch atop a Skyscraper, 1932"

response = client.models.generate_content(
    model="gemini-2.5-flash-image-preview",
    contents=[prompt, image],
)

for part in response.candidates[0].content.parts:
    if part.text is not None:
        print(part.text)
    elif part.inline_data is not None:
        image = Image.open(BytesIO(part.inline_data.data))
        image.save("lunch-restored.png")
```

Original and Output:

6) Working with Multiple Input Images

You can provide multiple images as input for more complex editing tasks.

```
from google import genai
from PIL import Image
from io import BytesIO

client = genai.Client(api_key="YOUR_API_KEY")

prompt = "Make the girl wear this t-shirt. Leave the background unchanged."

image1 = Image.open("girl.png")
image2 = Image.open("tshirt.png")

response = client.models.generate_content(
    model="gemini-2.5-flash-image-preview",
    contents=[prompt, image1, image2],
)

for part in response.candidates[0].content.parts:
    if part.text is not None:
        print(part.text)
    elif part.inline_data is not None:
        image = Image.open(BytesIO(part.inline_data.data))
        image.save("girl-with-tshirt.png")
```

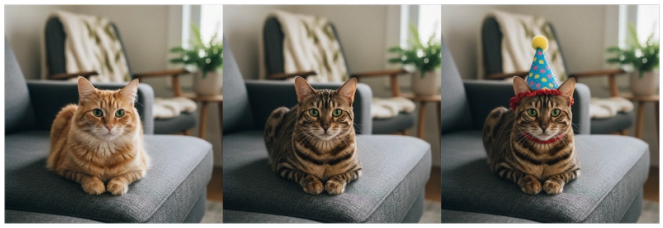
Inputs 1 & 2 and Output:

7) Conversational Image Editing

For iterative refinement, you can use a `chats` session to maintain context across multiple requests. This allows you to edit images conversationally.

```
from google import genai
from PIL import Image
from io import BytesIO

client = genai.Client(api_key="YOUR_API_KEY")

# Create a chat
chat = client.chats.create(
    model="gemini-2.5-flash-image-preview"
)

# Make the first image edit
response1 = chat.send_message(
    [
        "Change the cat to a bengal cat, leave everything else the same",
        Image.open("cat.png"),
    ]
)
# display / save image...

# Continue chatting and editing
response2 = chat.send_message("The cat should wear a funny party hat")
# display / save image...
```

Input and Outputs 1 & 2:



> **Tip**: If you notice image features starting to degrade or "drift" after many conversational edits, it's best to start a new session with the latest image and a more detailed, consolidated prompt to maintain high fidelity.

8) Best Practices and prompting tips for Nano Banana

To achieve the best results with Nano Banana, follow these prompting guidelines:

- **Be Hyper-Specific:** The more detail you provide about subjects, colors, lighting, and composition, the more control you have over the output.
- **Provide Context and Intent:** Explain the purpose or desired mood of the image. The model's understanding of context will influence its creative choices.
- **Iterate and Refine:** Don't expect perfection on the first try. Use the model's conversational ability to make incremental changes and refine your image.
- **Use Step-by-Step Instructions:** For complex scenes, break your prompt into a series of clear, sequential instructions.

- **Use Positive Framing:** Instead of negative prompts like "no cars," describe the desired scene positively: "an empty, deserted street with no signs of traffic."
- **Control the Camera:** Use photographic and cinematic terms to direct the composition, such as "wide-angle shot", "macro shot", or "low-angle perspective".

For a deeper dive into best practices, review the official blog post on prompting best practices and the prompting guide in the documentation.

9) Community Examples and Inspiration

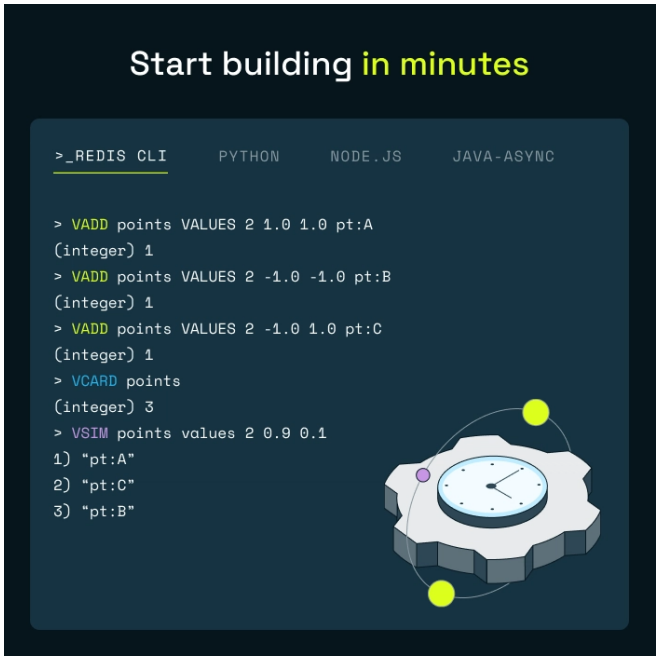
Explore what the community is building with Nano Banana:

10) Resources and Next Steps

This guide has covered the fundamentals of building with Nano Banana aka Gemini 2.5 Flash Image. You've learned how to set up your environment, generate and edit images, and apply advanced techniques. Now you're ready to start incorporating these powerful capabilities into your own projects.

For further reading, check out the official resources:

If you're building something cool with this, I'd love to see it! Feel free to DM or tag me on X: @patloeber.



49,000 responses in the Stack Overflow 2025 Developer Survey have spoken: Redis 8 is 2x faster, with Redis Query Engine and vector sets enabling real-time agent memory and semantic caching.

Learn more